

# A & B

# COMPUTING

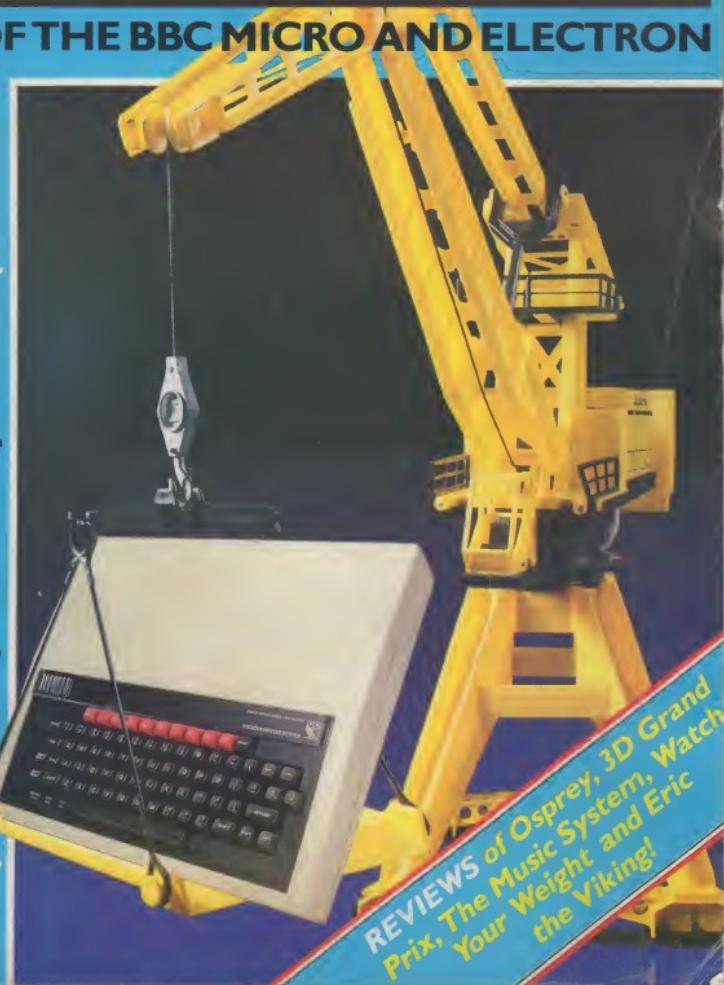
£1.25

FOR USERS OF THE BBC MICRO AND ELECTRON

**Take Control:**  
*Interfacing Techniques for User Port, RS423 and IEEE*

**Developing 6502 Ideas with Acorn's Own System**

**Plus:**  
*Desk Top Aids for Businessmen, Homing-In on Domestic Databases, Swopping Keys for Joysticks, and the Game of Life.*



REVIEWS of Osprey, 3D Grand Prix, The Music System, Watch Your Weight and Eric the Viking!

**Cover:** Pilot One's Computer Controlled Crane



**News** ..... 6

More of the latest add-ons, software releases and information to help you get the most from your computer.

**Mailsort** ..... 13

Your comments, complaints, ideas and opinions. Keep them coming.

**Dynamic Duo** ..... 16

An in-depth look at the facilities offered by Acorn's spreadsheet, ViewSheet, in combination with the Raven-20 board.

**Disc Menu** ..... 22

Create a menu of all your files on disc.

**Laser Cycles** ..... 28

A race against time. Vapourise the enemy before your cycle disintegrates.

**Random Access** ..... 34

Dave Carlos provides the answers to some of the questions he's often asked about discs and drives.

A&B Computing is constantly on the look-out for well-written articles and programs for publication. If you feel that your efforts meet our standards, please feel free to submit your work to us for consideration for publication.

All submitted material should be printed or typed, double spaced. Any programs submitted should be listed (55 character width emphasised if possible). A cassette of the program alone will not be considered. All programs must come complete with a full explanation of the operation, and where relevant, the structure. We also require the program in machine readable form (cassette, 40 track 5½", or 3" disc) plus any suitable screen photographs, printer dumps and so on.

All submissions will be acknowledged and the copyright in such works which will pass to Argus Specialist Publications Limited will be paid for at competitive rates. All work for consideration should be sent to the Editor at our Golden Square address.

## Volume Two Number

**Group Editor:** Elspeth Joiner

**Acting Editor:** Mark Webb

**Editorial Assistant:** Ione Holmes

**Software Assistant:** Simon Rockman

**Advertising Manager:** Barry Bingham

**Sales Executive:** Jonathan McGarry

**Chief Executive:** TJ Connell

**Accountant** ..... 36

How good is Acorn's book-keeping program?

**Life** ..... 40

An intriguing puzzle for the Beeb and Electron that will fascinate you for hours.

**Sorting On Index** ..... 45

Silversoft's disc-based database.

**CRTC Under Control** ..... 48

Direct Programming of the 6845 graphics chip. Apologies for the wait but it's well worth it! GOTO 48.

**Developing Ideas On 6502** ..... 55

A detailed look at Acornsoft's 6502 development system.

**ROM Report** ..... 63

Watford Electronics' ROM Manager reviewed.

**Buggy Briefing** ..... 65

The Economics Buggy becomes more versatile with a Penkit.

**Questions and Answers** ..... 68

Bruce Smith solves your computer problems.

**ROM Report** ..... 70

An analysis of the DUMPOUT 3 ROM from Watford Electronics.

**Solo Pilot** ..... 72

Pilot One's interface for the BBC teaches digital control.

**Bouncer** ..... 74

The second part of our series creates the sprites for this arcade game.

# A&B

## One January 1985

Published by Argus Specialist Publications Ltd., Number One, Golden Square, London W1R 3AB. Tel: 01 437 0626.

All work for consideration should be sent to the Editor of A&B Computing at our Golden Square address.



**Edsoft .....** 80

Educational programs for students of all ages.

**Create Your Own Adventure World .....** 90

Melvyn Wright's article concludes by explaining the importance of the story itself in a successful adventure game.



**Software Reviews .....** 92  
All the latest programs for the BBC and Electron.

**Down To Business .....** 104  
The Miracle WS2000, Pace Nightingale and Prism 1000 modems compared as executive aids.

**IEEE Experiment .....** 108  
Acorn's IEEE-488 interface is put through its paces.

**Software Listings .....** 112  
A guide to the multitude of programs available for your computer.

A&B Computing is published monthly on the first Friday of the month preceding cover date. Distributed by: SM Distribution Ltd, 16-18 Trinity Gardens, London SW9 8DX. Telephone: 01-274 8611. Printed in the UK by Garnett Print, Rotherham and London

The contents of this publication including all articles, designs, plans, drawings and programs and all copyright and other intellectual property rights therein belong to Argus Specialist Publications Ltd. All rights conferred by the Law of Copyright and other intellectual property rights and by virtue of international copyright conventions are specifically reserved to Argus Specialist Publications Ltd. Any reproduction requires the prior written consent of Argus Specialist Publications Ltd.

© Argus Specialist Publications Ltd 1984  
Printed in the U.K. by Garnett Print, Rotherham and London.

# Create Your Own Adventure World

Melvyn Wright

Last month I concentrated my attention on the desirable qualities of a good adventure master program. This month I will look at the second aspect of any adventure game, namely the story itself. The requirements here are much more difficult to define but they are just as important as the actual adventure program. You must place the player in an interesting environment, give him an objective to fulfil and provide a set of logical and inter-related puzzles for him to solve. You should also provide a reason for the player being in that situation that you have placed him, but this is not often done, and it doesn't seem to bother most people. They are not really interested in the past, all they want to know is how to escape in the future!

## OBJECTIVES

Broadly speaking, the objective of most adventure games falls into one or more of the following categories. The player has to a) collect various items of treasure, possibly storing them somewhere along the way; b) score a minimum number of points, usually achieved by solving all the puzzles; c) get to a certain location or find a certain object; d) kill a certain creature, or perform some heroic act, like rescuing a damsel in distress.

Many games incorporate more than one of these ideas, i.e. the player may have to get to a certain location in order to perform the heroic act, collecting and storing treasures along the way.

The setting for an adventure game is one of the first things that an adventure writer must decide upon when writing a new game. Many types of settings have been used but there is little doubt that the most popular type of environment, and the one favoured by the majority of adventurers, consists of a world which is as far removed from the present world as possible. Adventuring is a form of escapism and the player enjoys being the king of his own world in which he can forget the day-to-day troubles of reality.

This explains why the majority of adventures at present

**In the second of two articles, Melvyn Wright of EPIC Software explains how to ensure that your own adventure games are both exciting and playable.**

available take place either in a magical fantasy land, or in space. The average person would not normally have any expectations of visiting these kinds of environments during his lifetime, so the fascination is obvious. However, this does not mean that an adventure based upon the real world is a bad idea. When people get bored with their fantasy lands they may well want to escape back to reality!

## PUZZLING AND PLAUSIBLE

One of the most important aspects of any adventure game is the layout and type of the puzzles that have to be solved by the player in order to successfully complete the game. I would go as far as to say that it is these puzzles (and more importantly, their solutions) which will decide whether the game is to be a success or not. It does not matter how brilliant the program is, nor how fascinating the scenario, the player is likely to quickly become bored and frustrated if he is constantly confronted with tortuous problems which seem to have no logical solution.

On the other hand, an adventure game is like a jigsaw puzzle, inasmuch as once you have solved it, it becomes worthless. If someone spends £10 on a game which he then solves in a matter of days, he is very likely to feel cheated, and rightly so. Therefore, you must strike a balance when you are devising the puzzles for your adventure, and I shall devote the rest of the article to this very important aspect.

It is very difficult to judge exactly how hard your puzzles actually are when you are devising them, as you already know the answers! After a great deal of experience in this matter, I will warn you that the player will find them harder than you had intended when you devised them. Even experienced adventurers will get stuck at places where you didn't intend there to be any puzzles at all!

The golden rule is to make all the solutions logical. They can be as hard or easy as you think fit, but the solutions must be logical and plausible.

For instance, in a certain well-known adventure there is a portcullis which has to be raised to pass through. There is no indication given as to how this feat may be performed and the program does not respond to the usual efforts that would be required to carry out such a task, even if the player possesses objects that would appear to be useful. Consequently, most players give up in despair at this point.

The solution is in fact to rub the ruby, which is one of the items of treasure you have previously encountered. This solution is totally illogical and the only way of solving the problem would be by accident, or by cheating (as I did!). At no time is there any connection suggested between the ruby and the portcullis, nor are there any clues that would lead the player to think that the ruby had to be rubbed at some stage of the game. Whilst it is not unacceptable to arrange for the ruby to have magical powers to lift the portcullis, such a thing is so far

removed from reality that the total absence of clues is completely inexcusable. It would have been a simple matter to have said "the ruby vibrates slightly" when approaching the portcullis, or to arrange for the ruby to have a portcullis engraved upon it. If the writer wanted this to remain a particularly nasty problem then he could have had a red portcullis. This would have suggested an association between the two objects but would still have required a great deal of thought to solve the problem.

## GIVE US A CLUE

The previous example also demonstrates that it is not the puzzles, nor their solution which determine the difficulty of the game, it is the clues contained within the text of the messages that decide whether the game will be easy or hard. Even the most illogical puzzle becomes easy if its solution is virtually given away within the text of the game.

When writing your adventure, try to put yourself in the player's shoes and imagine what actions he is likely to attempt as he comes to each problem. If the solution does not suggest itself naturally, he must be given a prod in the right direction. Moreover, if a problem has to be solved in two stages, give him some encouragement when he has solved the first stage, otherwise he may give up altogether.

For instance, suppose you have a treasure chest that has to be opened by first unlocking it, then prising the lid free. The first attempt to open it should be met with the response "the chest is locked". When the player unlocks it and tries again the program should give a different message, i.e.: "the lid appears to be jammed". This combination of messages will steer the player gradually towards the full solution to the problem. Had he been confronted with the message: "you cannot do that" or "the chest will not open" then it is very unlikely that he would have realised that the puzzle needed a two-part solution, and would probably have given up after the first stage, not realising that he

had in fact solved half of it.

This takes us on to the subject of error messages. If a player attempts to do something which is clearly impossible, or downright silly, then he should expect to be told, simply: "You cannot do that." There is no need to use up valuable memory in trapping these errors and storing messages like: "you cannot take the door" or "you cannot eat the lamp". However, he should not be told: "you cannot do that" if he attempts to do something and is prevented from doing so because he has overlooked something.

In Epic games, if the player enters any command that the program recognises (even if it cannot yet be carried out), the player is given a special message depending upon the reason why the command cannot yet be executed. For example, he may be in the wrong place, or in the right place but at the wrong time.

It is often a good idea to include a blanket error message for each of the more common verbs in the game, for example: "you cannot eat that" or "you cannot kill anything here", but beware, this can cause confusion. If the player mis-spells the noun and types: "EAT BISIT" he will be told "you cannot eat that" and may not spot his error, as the program appears to understand him. If he had been told "you cannot do that" the lack of recognition by the program would probably have caused him to re-examine this command. So avoid trying to be too clever with your error messages, keep them as general as possible and make sure that they sound right whatever the player types in.

Although it is part of the game to work out the vocabulary of the program, you should never enforce a strict input from the player for the solution of any of your puzzles (except passwords). You must always try to work out what the player is likely to type in a given situation then cater for it in your program. It is a poor game if the player has to constantly keep rephrasing his commands to get the program to understand him, especially if he is typing the obvious. He will even-



tually give up and assume that he has the wrong solution. For example, if he has a boat the player should be able to cross a river by typing: "CROSS RIVER", "BOARD BOAT", "ROW BOAT", etc. Don't provide "EMBARK" as the only accepted command.

## LOGICAL MAPPING

I have stressed the importance of logic when drawing up your puzzles and this applies equally to the map of your adventure layout. If the player ventures north from a certain location, going south should return him to his original location, unless it is a one-way route. He should not end up at a third location simply by typing two opposite directions. Furthermore, if it is a one-way route it should be made clear why it is not possible to return, e.g. "you have jumped off the cliff" or "the door slams behind you".

Using illogical directions makes it impossible for the adventurer to make a map of his journey and whilst you may think

this is good fun, experience has shown that the majority of players are infuriated by this type of maze. It is quite possible to construct a complicated maze without having to resort to illogical paths which are not retracable. The use of these reduces the game to a test of trial and error which can only be solved by accident. This is not what adventuring is all about. The provision of this type of maze could be justified if a novel method of solving it is devised, as is the case in some of the Acornsoft adventures.

Another illogical trick used by some adventure writers is to kill off the player at random. The designer will arrange for there to be a monster prowling one section of the game and if you are in that section when the wrong random number comes up — end of game. Thus the section is reduced to a test of patience while you constantly save your position at every move, and reload it each time you get killed. However, this trick is permissible if there is a valid way of tending off the monster, as long as a logical association is established between the monster and the required

weapon beforehand. However, it is acceptable to kill the player suddenly if you clearly explain the reasons for him being killed. In this case he can avoid it next time.

Although the subject of mapping has already been discussed, there are a few additional details which can be incorporated to make the game more playable.

Make it easy for the player to move about and explore his environment, particularly at the start of the game. Do not use 45 degree compass points, stick to north, south, east, west, and occasionally up and down. These make exploration much easier, and are more convenient to type in. Finally, do not kill the player simply for moving about in the dark, unless there is a good reason which is explained in the text.

You must decide for yourself exactly how easy or difficult your adventure game is going to be. However, by implementing the techniques that I have outlined in this article, you will ensure that your game remains interesting and playable, whatever the difficulty.

# IEEE Experiment

David Abbott

## WHAT IS THE IEEE-488 INTERFACE BUS?

Also known as the General Purpose Interface Bus (GPIB), the Hewlett-Packard Interface Bus (HPIB), and the International Electrotechnical Commission (IEC) 625 bus, the Institute of Electrical and Electronic Engineers (IEEE) standard number 488 defines a 'standard digital interface for programmable instrumentation'. Originally written in 1975 and revised in 1978, the standard deals with systems that use byte-serial, bit-parallel means to transfer digital data among a group of instruments and system components.

Put simply this means that data is transmitted over the interface a byte at a time (byte-serial), over eight separate data lines (bit parallel). The flow of data is controlled by three command lines which are used to indicate the successful passage of data from one instrument to another. This is known as a three-wire handshake. In all there are 24 lines connecting instruments on the bus, comprising eight data lines, eight ground lines and eight command lines, three of which are the handshake lines, the other four handling general interface management.

There are two types of data transmitted on the bus: data used to manage the interface itself, and data used by the devices connected by the bus. In the IEEE-488 standard these are known as interface messages and device dependent messages respectively. Devices connected by the bus have one or more of the capabilities of listening, talking and controlling. Under the control of interface messages a listener can receive device dependent messages from another device on the bus, and a talker can send device dependent messages to another device on the bus. A controller sends the interface messages that command the specific actions in the other devices. An example of a listener is a power supply which receives information on a voltage setting



## Exhaustive testing for Acorn's IEEE.

and acts upon it. An example of a talker is a digital thermometer which, when commanded, transmits its current temperature reading. A digital multimeter has both the ability to listen and to talk, and a computer usually has all three abilities of listening, talking and controlling.

## THE ACORN IEEE-488 INTERFACE

The Acorn interface was designed for Acorn by a company called Intelligent Interfaces Ltd, who also manufacture the unit. The interface supplied for review was, unlike a lot of review hardware, a production unit, working to the full specification, and containing complete documentation. It arrived

in an expanded polystyrene box with one side thoughtfully marked 'TOP'. Inside, in addition to the interface, was an EPROM containing the IEEE filing system, a ribbon cable for connecting the interface to an instrument and the user guide, also produced by Intelligent Interfaces. Also supplied with the review hardware, though perhaps not standard issue, was an application note written by Intelligent Interfaces on the use of functions and procedures in programming the interface in Basic.

The electronics of the interface are housed in the standard Acorn box (teletext adaptor, second processor etc), and connect to the 1MHz bus by a 34 way ribbon cable. Being only 50cm long the cable dictates that the interface is placed next to the microcomputer. The ribbon cable

is permanently attached to the interface, but has a 34 way header to allow connection of further 1MHz bus peripherals at the same time as the interface. The interface does not derive its power supply from the computer but has its own mains connection. In the same style as the computer, the interface comes with an integral mains lead with moulded three-pin plug and on/off switch on the rear panel.

On opening the interface one is struck by how little there is inside. The electronics, nine integrated circuits and a handful of passive components, are on a single printed circuit board, and the largest single component is the mains transformer. The construction is excellent. The three main integrated circuits are socketed, and the board does not contain any modifications. By using shrouded connectors on all the a.c. leads, it is impossible to touch the mains supply anywhere.

The user guide also reflects similar attention to detail. A spirally bound A5 volume of 64 pages and appendices, it is clearly laid out and properly printed. It contains introductory information to get the user going, then goes into a deeper study of the interface operation. A complete list of the IEEE commands is given, with programming examples and a clear description of the command and its effect. The guide includes a short description on using the filing system in assembler, and gives the structure of the OSWORD block for each command. There are appendices covering the minimum abbreviations for each command, error messages, and the fitting of the filing system EPROM. The guide finishes with a single page index.

Despite recommending at the beginning of the user guide that the IEEEFS EPROM be fitted by a dealer, appendix 5 gives very detailed instructions on how to fit the EPROM yourself. As with the hardware, the EPROM supplied contained the current production filing system, version 0.02. Given Acorn's penchant for multiple issues of filing systems (eg Econet), it was gratifying to find from a phone call to



Intelligent Interfaces that this was the definitive version despite the 'temporary' sound of the version number (no OS 0.10 here). On installing the EPROM and typing \*HELP IEEE one is greeted by the rather curt 'IEEE Filing system 0.02', 'Please refer to the User Guide', not the usual list of available commands. As the IEEE commands are not issued directly, prefixed by an asterisk as in the DFS, perhaps this omission is not surprising.

## USING THE ACORN INTERFACE

Having installed the EPROM and connected the interface to the 1MHz bus everything was ready to go. Starting at the simplest end I chose to control a single listen-only device, a power supply, which was connected to the interface with the supplied cable.

Communication between the computer and the interface is through two channels. A command channel is used for transmitting bus commands and for receiving information on the interface status, and a data channel used for transmitting and receiving data to and from other devices on the bus. This rather unusual approach is a consequence of the communication being through a filing system rather than a more conventional I/O driver.

The first steps therefore are to select the filing system and to open the command and data channels:

```
10 * IEEE
20 cmd% = OPENIN
("COMMAND")
30 data% = OPENIN
("DATA")
```

The integer variables used in lines 20 and 30 can be chosen to suit your needs, but having been assigned, all transfer of information is referenced by these variables.

The bus address of the power supply is assigned in a similar manner:

```
40 power% = OPENIN("5")
```

where '5' is the numeric address set up on dual-in-line switch on the rear of the power supply. All devices on the bus have a numeric address which is used to uniquely identify them. The computer must also be assigned an address. Any number between 0 and 30 is acceptable, though custom has the system controller assigned to address '0'.

```
50 PRINT #cmd%, "BBC
DEVICE NO", 0
```

This instruction is sent to the interface through the command channel referenced by the integer variable 'cmd%', assigned in line 20. The phrase "BBC DEVICE NO" is one of the 28 commands that may be issued to the interface. It can be typed in full or abbreviated to 'B'. All commands have abbreviated forms with the most used commands abbreviated to a single letter. The saving in memory space comes with a lack of readability, a problem experienced in normal basic operation when using long variable names.

Having assigned addresses the interface is initialised with the CLEAR command, and instruments prepared for remote operation with the REMOTE ENABLE command:

```
60 PRINT      # cmd%,
"CLEAR"
70 PRINT      # cmd%,
"REMOTE ENABLE"
```

Everything is nearly ready now for the power supply to be given an instruction. This will be a string of ASCII characters. In order for the device to know

when the string has finished it recognises particular characters as marking the end of the string. These characters may change from one instrument to another and should be set up if they differ from the default character of 'line feed'. The power supply used recognised 'carriage return/line feed', so:

```
80 PRINT #cmd%, "END OF
STRING", CHR$(13) +
CHR$(10)
```

This finishes the preparation. It seems awkward, but only needs to be done once, so can be relegated to a procedure:

```
10 * IEEE
20 PROCinitialise
```

This gives readability to the main program flow, and defines a procedure that could be used in subsequent programs.

The power supply used had two ranges, 0-10V, and 0-50V, with 1000 discrete programmable steps in each range. The required voltage had to be translated to a particular step in the 0-999 range and prefixed with '1' for the lower range, and '2' for the higher range, giving an effective four digit number between 1000 and 2999. This number is then transmitted to the power supply as a string.

A combination of function and procedure was used to get round the problem of a procedure not returning a value:

```
30 REPEAT
40 voltage = FNget_vols
50 PROCVolts (voltage)
60 UNTIL FALSE
```

The function to obtain the required voltage could be a simple INPUT statement, or could include limit value checking, to ensure that the requested voltage lay between 0 and 50 volts. The procedure to set the voltage has to scale the voltage within the required range, convert it to a string, and transmit it to the interface:

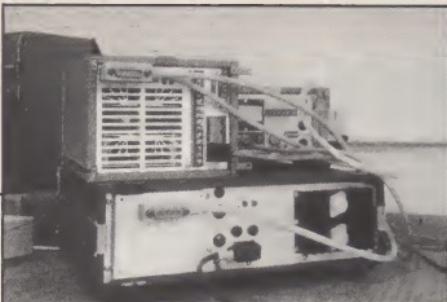
```
100 DEFPROCvolts
(voltage)
110 IF VOLTAGE > 9.990
THEN volts$ = STR$(2000 + INT((voltage / 0.05 + 0.5)))
ELSE volts$ = STR$(1000 + INT((voltage / 0.01 + 0.5)))
120 PRINT # cmd%,
"LISTEN", power%,
"EXECUTE"
130 PRINT # data%, volts%
140 PRINT # cmd%,
"UNLISTEN"
150 ENDPROC
```

When these lines are executed the power supply that has been assigned to 'power%' sets itself to the voltage 'voltage'. This may seem very long winded, and is compared with other dedicated controllers, but by using the function and procedure statements available on the BBC computer, a concise and readable main program can still be maintained.

## GETTING IN DEEPER

Having mastered the control of a simple listen-only device, the

**CONTINUED OVER**



next step was to link in a talker so that information could be received by the computer. The obvious complement to the power supply was a volt-meter connected up to measure the voltage set upon the power supply.

As the interface has already been set up the only preparatory statement necessary is to assign the address of the meter in PROC initialisation:

**80 dvm% = OPENIN ("7")**

The taking of a measurement is a two stage process. Firstly the meter has to be set to measure the correct function, volts in this case, with a given resolution. More complex meters might require more information such as which input on a multi-input instrument, whether a number of readings are to be taken and averaged, is any processing of the reading necessary before outputting the data etc. Information on the codes required are obtained from a particular instruments handbook.

For the meter in question the setting up was accomplished by:

```
200 PRINT # cmd%,  
    "LISTEN", dvm%, "EX-  
    ECUTE"  
210 PRINT # data%,  
    "J3D3T1N1Y1Q1R0"  
220 PRINT # cmd%,  
    "TRIGGER", dvm%,  
    "EXECUTE"  
230 PRINT # cmd%,  
    "UNLISTEN"
```

This sequence only needs to be executed once unless any of the parameters in line 210 need to be changed, so it can normally be included in the initialisation procedure.

As actually reading the meter needs the return of the measured value, this is handled by a function call:

So far we have only seen 10 of the possible 28 commands in use. The remaining 18 fall into three groups:

### 90 answer = FNmeter

```
.  
300 DEFFNmeter  
310 PRINT # cmd%,  
    "TALK", dvm%  
320 INPUT # data%,  
    voltage$  
330 PRINT # cmd%, "UN-  
    TALK"  
340 = VAL (voltage$)
```

The way the transfer works should be fairly obvious from this function. A particular instrument, here denoted by 'dvm%' is commanded to talk, ie transmit its data. This is inputted on the data channel into the variable 'voltage\$'. The command 'UNTALK' puts the bus back into a quiescent state, and the value transmitted is turned from a string to a numeric value using the function 'VAL' and returned to the calling program.

## MORE ADVANCED USES

The simple examples of the previous section where a talk-only device, the power supply, was set to a given voltage, and a talker/listener, the voltmeter, was set to a known state and made to transmit a measured value, may seem trivial, but in essence such sequences are all that is necessary in perhaps 80% of applications. Transmission of simple strings between instruments and the controller is all that is necessary to measure the frequency response of an amplifier, to measure the frequency spectrum of an oscillator, or measure the performance of a complex electronic system.

So far we have only seen 10 of the possible 28 commands in use. The remaining 18 fall into three groups:

- 1 — Block data transfer
- 2 — Instrument status
- 3 — Bus status control

The block data transfer instructions 'READ BINARY', 'WRITE BINARY', and 'TRANSFER' are for moving specified numbers of bytes both between a controller and an instrument and between two instruments—one acting as the talker, the other as the listener. In this way data dumps may be performed by copying the contents of a instrument memory straight to a plotter.

The second group, instrument status, covers the use of serial and parallel polls and bus status. The status request command 'STATUS' returns an integer's worth, ie four bytes, of information on the bus status. Individual bits indicate the current status of a number of bus functions. To check for specific states the requisite bit has to be masked out. For instance if an instrument needs attention from the controller, it generates a service request (SRQ). The presence of an SRQ is indicated by bit 5 in the status word being set. This could be handled by:

```
490 DEFPROCstatus  
500 PRINT # cmd%,  
    "STATUS"  
510 INPUT # cmd%, state%  
520 state% = state% AND  
    &20  
530 IF state% = 0 ENDPROC
```

If SRQ is not set the status procedure terminates, otherwise the SRQ must be handled. How to handle the SRQ brings in the serial and parallel poll instructions. If there is only one instrument on the bus then it must be generating the SRQ and it can be accessed directly using a serial poll. This instructs the instrument to return status information to the controller, from which the controller can work out why the SRQ was generated and take appropriate action.

If there is more than one instrument on the bus, then which one has generated the SRQ is not known. Two courses of action are then open to the controller. Either it can serial poll each in-

strument in turn until the one generating the SRQ is located, or get the instrument to 'own-up' by using a parallel poll. In a parallel poll each instrument is allocated a particular bit in a status byte. When a parallel poll is issued the instrument generating the SRQ sets its bit in the status byte, thus the controller can immediately distinguish which of eight instruments (there are only eight bits in a byte), is requesting attention. This particular instrument may then be serial polled for more specific information. This sequence is illustrated in the following code:

```
540 PRINT # cmd%,  
    "PARALLEL POLL RE-  
    QUEST"  
550 INPUT # cmd%, poll%  
560 IF poll% AND 4 PRO-  
    Cserial_poll_dvm  
570 IF poll% AND 6 PRO-  
    Cserial_poll_psu  
580 IF poll% AND 8 PRO-  
    Cserial_poll_psu  
580 ENDPROC  
590 DEFPROC  
    serial_poll_dvm  
600 PRINT # cmd%,  
    "SERIAL POLL",  
    dvm%, 1  
610 INPUT # cmd%,  
    status$  
620 status = ASC (status$)  
630 IF status AND &40 = 0  
    ENDPROC  
640 status = status AND 7  
650 IF status = 4 PRINT  
    "Out of range value"  
660 IF status = 5 PRINT  
    "Unrecognised  
    character"  
670 ENDPROC
```

The digital voltmeter used was set to respond to a parallel poll on bit 3 (line 560). When presented with a serial poll command it returns one byte of status information (line 600), bit 6 of which is set if it generated the SRQ (line 630). This check is merely a 'belt-and-braces' way of double checking the origin of the SRQ. If it did generate the SRQ, bits 0, 1, and 2 are set to indicate one of two error conditions, these are examined on lines 640-660 and appropriate messages printed.

Associated commands not

used in this example are "PARALLEL POLL ENABLE" and "PARALLEL POLL DISABLE" which enable/disable the parallel poll on a specific addressed instrument, and "PARALLEL POLL UNCONFIGURE" which stops all instruments taking part in a parallel poll.

In large systems having more than eight polling instruments, there may be two instruments responding on one bit of a parallel poll. In this case it might be necessary for the controller to serial poll each to find the instrument requesting service.

The third and final group of commands are those associated with general bus status control. There are four pairs of commands handling similar functions, and a single un-paired command:

**GO TO LOCAL** — return addressed device to local

**REMOTE DISABLE** — return all devices to local

**SELECTED DEVICE CLEAR** — clears specified device

**DEVICE CLEAR** — clears all devices on the bus

**TAKE CONTROL** — pass control to another controller

**REQUEST CONTROL** — request control from system controller

**TIME ON** — enable timeout

**TIME OFF** — disable timeout

**LOCAL LOCKOUT** — locks panel controls on a specified device

These are all self-explanatory and are used in a variety of ways. For instance 'GO TO LOCAL' would enable an adjustment to an instrument to be made, setting a power supply voltage, whilst other instruments were still under bus control. 'REMOTE DISABLE' on the other hand might be the last statement in a program, that returns all instruments to local control before halting.

Not all instruments respond to all commands. A careful examination of the operating manual will reveal which do not respond to polling, or do not permit block transfer of data. The set of 28 commands, though, allows as much control as each instrument permits, giving the programmer the opportunity to extract as much from the instrument as possible.

## IN CONCLUSION

On the hardware side the interface was virtually faultless. Its construction is sound with several nice touches, such as the covering of all points at mains voltage inside the interface. With so few components the reliability should be good, but if anything does go wrong then maintenance should be straightforward due to the ease of access.

My only quibble on the hardware side is with the cable connecting the interface to the instruments. The usual configuration is for a screened cable with a stackable connector at each end.

The cable supplied by Acorn is a ribbon cable with a non-stackable connector at each end. This forces the controller to be at the end of a chain, or a single branch in a star-connected system which is a limitation not experienced with other controllers. In addition the non-stackable connector does not have the screw-fit attachment found on conventional bus connectors, leaving the possibility of it becoming disconnected behind a bank of equipment.

When it comes to the software, different people expect different things. The command set in EPROM enables you to perform quite complex bus control operations, but in a rather laborious fashion. The need to operate at the 'TALK/UNTALK' level is balanced by the ability to confine these sorts of operation to functions and procedures, in which case the main program flow takes on the look of a higher level controller.

Documentation, though not abundant, is sufficient, and what there is is well written and well presented. It is thought aimed more at the experienced user than at the beginner. A novice in programming instruments on the bus will need more information than the user guide provides, particularly as the low level approach to control means that the

programmer is not protected by a sophisticated I/O driver.

I experienced no problems with the interface I designed in the EPROM, connected up the 1-MHz bus and away it went. However another interface, purchased by a colleague, failed to work until a pull-up resistor was connected to pin 1 of the EPROM. Talking both to Intelligent Interfaces and to Acorn failed to throw any real light on the cause, as it seemed to be an isolated incident that no-one else had reported.

Whilst not exactly a problem, I did come across an 'effect' which was a little disconcerting. In common with all filing system ROMS, the IEEEFS moves PAGE upwards in memory to claim space for a work area. If this space is re-claimed for program use it can still be affected by the IEEEFS. Pressing ESCAPE or BREAK/OLD causes certain memory locations to be overwritten, resulting in a corrupted program. This could be embarrassing if one was trying to run an analysis program on data collected over the interface, where all memory space was valuable. This effect is due to the necessity to leave the bus in a known state if a control program was interrupted using the ESCAPE or BREAK keys. As the claiming of work space, and the overwriting of these memory locations takes place even if the interface is not connected (cf the teletext adaptor), then the only way out is to remove the EPROM. Not a practical proposition on a day-to-day basis though.

On balance I found the interface to be easy to use. As an experienced user of the bus, the low level programming required was not a problem, though a first time user would require more general information on the bus operation than had been supplied.

For use in industry or in education, the Acorn interface provides a cost effective alternative to the currently acceptable range of controllers. Question marks fall over reliability, maintenance and support, but the answers will only be found with time.

